

# On the Query Complexity of Black-Peg AB-Mastermind

Mourad El Ouali Christian Glazik Volkmar Sauerland and Anand Srivastav

Department of Computer Science  
Christian-Albrechts-Universität zu Kiel, Kiel, Germany  
<cgl,meo,vsa,asr>@informatik.uni-kiel.de

## Abstract

*Mastermind* game is a two players zero sum game of imperfect information. The first player, called “codemaker”, chooses a secret code and the second player, called “codebreaker”, tries to break the secret code by making as few guesses as possible, exploiting information that is given by the codemaker after each guess. In this paper, we consider the so called Black-Peg variant of Mastermind, where the only information concerning a guess is the number of positions in which the guess coincides with the secret code. More precisely, we deal with a special version of the Black-Peg game with  $n$  holes and  $k \geq n$  colors where no repetition of colors is allowed. We present upper and lower bounds on the number of guesses necessary to break the secret code. We first come back to the upper bound results introduced by El Ouali and Sauerland (2013). For the case  $k = n$  the secret code can be algorithmically identified within less than  $(n - 3)\lceil \log_2 n \rceil + \frac{5}{2}n$  queries. That result improves the result of Ker-I Ko and Shia-Chung Teng (1985) by almost a factor of 2. For the case  $k > n$  we prove an upper bound for the problem of  $(n - 2)\lceil \log_2 n \rceil + k + 1$ . Furthermore we prove a new lower bound for (a generalization of) the case  $k = n$  that improves the recent result of Berger et al. (2016) from  $n - \log \log(n)$  to  $n$ . We also give a lower bound of  $k$  queries for the case  $k > n$ .

**Keywords:** Mastermind; combinatorial problems; permutations; algorithms

## 1 Introduction

In this paper we deal with *Mastermind*, which is a popular board game that in the past three decades has become interesting from the algorithmic point of view. Mastermind is a two players board game invented in 1970 by the postmaster and telecommunication expert Mordecai Meirowitz. The idea of the game is that the codemaker chooses a secret color combination of  $n$  pegs from  $k$  possible colors and the codebreaker has to identify the code by a sequence of queries and corresponding information that is provided by the codemaker. All queries are also color combinations of  $n$  pegs. Information is given about the number of correctly positioned colors and further correct colors, respectively. Mathematically, the codemaker selects a vector  $y \in [k]^n$  and the codebreaker gives in each iteration a query in form of a vector  $x \in [k]^n$ . The codemaker replies with a pair of two numbers, called  $\text{black}(x, y)$  and  $\text{white}(x, y)$ , respectively. The first one is the number of positions in which both vectors  $x$  and  $y$  coincide and the second one is the number of additional pegs with a right color but a wrong position:

$$\begin{aligned} \text{black}(x, y) &= |\{i \in [n]; x(i) = y(i)\}|, \\ \text{white}(x, y) &= \max_{\sigma \in S_n} |\{i \in [n]; y(i) = x(\sigma(i))\}| \\ &\quad - \text{black}(x, y). \end{aligned}$$

The Black-Peg game is a special version of Mastermind, where answers are provided by black information, only. A further version is the so-called AB game in which all colors within a code must be distinct. In this paper, we deal with a special combination of the Black-Peg game and the AB game, where both the secret vector and the guesses must be composed of pairwise distinct colors ( $k \geq n$ ) and the answers are given by the black information, only.

**Related Works:** In 1963, several years before the invention of Mastermind as a commercial board game, Erdős and Rényi [7] analyzed the same problem with two colors. One of the earliest analysis of this game after its commercialization dealing with the case of 4 pegs and 6 colors was done by Knuth [17]. He presented a strategy that identifies the secret code in at most 5 guesses. Ever since the work of Knuth the general case of arbitrary many pegs and colors has been intensively investigated in combinatorics and computer science literature. In the field of complexity, Stuckman and Zhang [21] showed that it is  $\mathcal{NP}$ -complete to determine if a sequence of queries and answers is satisfiable. Concerning the approximation aspect, there are many works regarding different methods [2, 3, 4, 6, 9, 10, 11, 12, 14, 19, 20, 21]. The Black-Peg game was first introduced by Chvátal for the case  $k = n$ . He gave a deterministic adaptive strategy that uses  $2n\lceil\log_2 k\rceil + 4n$  guesses. Later, Goodrich [13] improved the result of Chvátal for arbitrary  $n$  and  $k$  to  $n\lceil\log_2 k\rceil + \lceil(2 - 1/k)n\rceil + k$  guesses. Moreover, he proved in the same paper that this kind of game is  $\mathcal{NP}$ -complete. A further improvement to  $n\lceil\log_2 n\rceil + k - n + 1$  for  $k > n$  and  $n\lceil\log_2 n\rceil + k$  for  $k \leq n$  was done by Jäger and Peczarski [15]. Recently, Doerr et al. [5] improved the result obtained by Chvátal to  $\mathcal{O}(n \log \log n)$  and also showed that this asymptotic order even holds for up to  $n^2 \log \log n$  colors, if both black and white information is allowed. For the AB game Jäger and Peczarski [16] proofed exact worst-case numbers of guesses for fixed  $n \in \{2, 3, 4\}$  and arbitrary  $k$ . Concerning the combination of both variants, Black-Peg game and AB game, for almost 3 decades the work due to Ker-I Ko and Shia-Chung Teng [18] was the only contribution that provides an upper bound for the case  $k = n$ . They presented a strategy that identifies the secret permutation in at most  $2n \log_2 n + 7n$  guesses and proved that the corresponding counting problem is  $\#\mathcal{P}$ -complete.

**Our Contribution:** In this paper we consider the Black-Peg game without color repetition. We first present a polynomial-time algorithm that identifies the secret permutation in less than  $n \log_2 n + \frac{3}{2}n$  queries in the case  $k = n$  and in less than  $n \log_2 n + k + 2n$  queries in the case  $k > n$ . The constructive strategy originated in the work of El Ouali and Sauerland [8]. Our result for the case  $k = n$  improves the result of Ker-I Ko and Shia-Chung Teng [18] by almost a factor of 2. Furthermore we analyze the worst-case performance of query strategies for both variants of the Game and give a new lower bound of  $n$  queries for the case  $k = n$ , which improves the recently presented lower bound of  $n - \log \log(n)$  by Berger et al [1]. We note, however, that the corresponding asymptotic bound of  $\mathcal{O}(n)$  is long-established. For  $k > n$  we give a lower bound of  $k$ . Both lower bounds even hold if the codebreaker is allowed to use repeated colors in his guesses.

## 2 Upper Bounds on the Number of Queries

We first consider Black-Peg Mastermind with  $k = n$  and the demand for pairwise distinct colors in both the secret code and all queries, i.e., we deal with permutations in  $S_n$ .

### 2.1 The Case $k = n$ : Permutation-Mastermind

For convenience, we will use the term permutation for both, a mapping in  $S_n$  and its one-line representation as a vector. Our algorithm for finding the secret permutation  $y \in S_n$  includes two main phases which are based on two ideas. In the first phase the codebreaker guesses an initial sequence of  $n$  permutations that has a predefined structure. In the second phase, the structure of the initial sequence and the corresponding information by the codemaker enable us to identify correct *components*  $y_i$  of the secret code one after

another, each by using a binary search. Recall, that for two codes  $w = (w_1, \dots, w_n)$  and  $x = (x_1, \dots, x_n)$ , we denote by  $\text{black}(w, x)$  the number  $|\{i \in [n] \mid w_i = x_i\}|$  of components in which  $w$  and  $x$  are equal. We denote the mapping  $x$  restricted to the set  $\{s, \dots, l\}$  with  $(x_i)_{i=s}^l$ ,  $s, l \in [n]$ .

**Phase 1.** Consider the  $n$  permutations,  $\sigma^1, \dots, \sigma^n$ , that are defined as follows:  $\sigma^1$  corresponds to the identity map and for  $j \in [n-1]$ , we obtain  $\sigma^{j+1}$  from  $\sigma^j$  by a circular shift to the right. For example, if  $n = 4$ , we have  $\sigma^1 = (1, 2, 3, 4)$ ,  $\sigma^2 = (4, 1, 2, 3)$ ,  $\sigma^3 = (3, 4, 1, 2)$  and  $\sigma^4 = (2, 3, 4, 1)$ . Within those  $n$  permutations, every color appears exactly once at every position and, thus, we have

$$\sum_{j=1}^n \text{black}(\sigma^j, y) = n. \quad (1)$$

The codebreaker guesses  $\sigma^1, \dots, \sigma^{n-1}$  and obtains the additional information  $\text{black}(\sigma^n, y)$  from (1).

**Phase 2.** The strategy of the second phase identifies the values of  $y$  one after another. This is done by using two binary search routines, called `FINDFIRST` and `FINDNEXT`, respectively. The idea behind both binary search routines is to exploit the information that for  $1 \leq i, j \leq n-1$  we have  $\sigma_i^j = \sigma_{i+1}^{j+1}$ ,  $\sigma_i^n = \sigma_{i+1}^1$ ,  $\sigma_n^j = \sigma_1^{j+1}$  and  $\sigma_n^n = \sigma_1^1$ . While, except for an unfrequent special case, `FINDFIRST` is used to identify the first correct component of the secret code, `FINDNEXT` identifies the remaining components in the main loop of the algorithm. Actually, `FINDFIRST` would also be able to find the remaining components but requires more guesses than `FINDNEXT` (twice as many in the worst case). On the other hand, `FINDNEXT` only works if at least one value of  $y$  is already known such that we have to identify the value of one secret code component in advance.

**Identifying the First Component:** Equation (1) implies that either  $\text{black}(\sigma^j, y) = 1$  holds for all  $j \in [n]$  or that we can find a  $j \in [n]$  with  $\text{black}(\sigma^j, y) = 0$ .

In the first case, which is unfrequent, we can find one correct value of  $y$  by guessing at most  $\frac{n}{2} + 1$  modified versions of some initial guess, say  $\sigma^1$ . Namely, if we define a guess  $\sigma$  by swapping a pair of components of  $\sigma^1$ , we will obtain  $\text{black}(\sigma, y) = 0$ , if and only if one of the swapped components has the correct value in  $\sigma^1$ .

In the frequent second case, we find the first component by `FINDFIRST` in at most  $2\lceil \log_2 n \rceil$  guesses. The routine `FINDFIRST` is outlined as Algorithm 1 and works as follows: In the given case, we can either find a  $j \in [n-1]$  with  $\text{black}(\sigma^j, y) > 0$  but  $\text{black}(\sigma^{j+1}, y) = 0$  and set  $r := j+1$ , or we have  $\text{black}(\sigma^n, y) > 0$  but  $\text{black}(\sigma^1, y) = 0$  and set  $j := n$  and  $r := 1$ . We call such an index  $j$  an *active* index. Now, for every  $l \in \{2, 3, \dots, n\}$  we define the code

$$\sigma^{j,l} := \left( (\sigma_i^j)_{i=1}^{l-1}, \sigma_1^r, (\sigma_i^r)_{i=l+1}^n \right),$$

and call the peg at position  $l$  in  $\sigma^{j,l}$  the pivot peg. From the information  $\sigma_i^j = \sigma_{i+1}^r$  for  $1 \leq i \leq n-1$  we conclude that  $\sigma^{j,l}$  is actually a new permutation as required. The fact that  $\text{black}(\sigma^r, y) = 0$  implies that the number of correct pegs up to position  $l-1$  in  $\sigma^j$  is either  $\text{black}(\sigma^{j,l}, y)$  (if  $y_l \neq \sigma_1^r$ ) or  $\text{black}(\sigma^{j,l}, y) - 1$  (if  $y_l = \sigma_1^r$ ). For our algorithm, we will only need to know if there exist one correct peg in  $\sigma^j$  up to position  $l-1$ . The question is cleared up, if  $\text{black}(\sigma^{j,l}, y) \neq 1$ . On the other hand, if  $\text{black}(\sigma^{j,l}, y) = 1$ , we can define a new guess  $\rho^{j,l}$  by swapping the pivot peg with a wrong peg in  $\sigma^{j,l}$ . We define

$$\rho^{j,l} := \begin{cases} \left( (\sigma_i^j)_{i=1}^l, \sigma_1^r, (\sigma_i^r)_{i=l+2}^n \right) & \text{if } l < n \\ \left( \sigma_1^r, (\sigma_i^j)_{i=2}^{n-1}, \sigma_1^j \right) & \text{if } l = n \end{cases}$$

assuming for the case  $l = n$ , that we know that  $\sigma_1^j \neq y_1$ . We will obtain  $\text{black}(\rho^{j,l}, y) > 0$ , if and only if the pivot peg had a wrong color before, meaning that there is one correct peg in  $\sigma^j$  in the first  $l-1$  places. Thus, we can find the position  $m$  of the left most correct peg in  $\sigma^j$  by a binary search as outlined in Algorithm 1.

---

**Algorithm 1:** Function FINDFIRST

---

```
input : Code  $y$  and an active index  $j \in [n]$ 
output: Left most correct peg position in  $\sigma^j$ 
1 if  $j = n$  then  $r := 1$  ;
2 else  $r := j + 1$ ;
3  $a := 1$ ;
4  $b := n$ ;
5  $m := n$  ; // position to be found
6 while  $b > a$  do
7    $l := \lceil \frac{a+b}{2} \rceil$  ; // pivot position
8   Guess  $\sigma^{j,l} := ((\sigma_i^j)_{i=1}^{l-1}, \sigma_1^r, (\sigma_i^r)_{i=l+1}^n)$ ;
9    $s := \text{black}(\sigma^{j,l}, y)$ ;
10  if  $s = 1$  then
11    if  $l < n$  then  $\rho^{j,l} := ((\sigma_i^j)_{i=1}^l, \sigma_1^r, (\sigma_i^r)_{i=l+2}^n)$ ;
12    else  $\rho^{j,l} := (\sigma_1^r, (\sigma_i^j)_{i=2}^{n-1}, \sigma_1^j)$ ;
13    Guess  $\rho^{j,l}$ ;
14     $s := \text{black}(\rho^{j,l}, y)$ ;
15  if  $s > 0$  then
16     $b := l - 1$ ;
17    if  $b < m$  then  $m := b$ ;
18  else  $a := l$ ;
19 Return  $m$ ;
```

---

**Identifying a Further Component:** For the implementation of FINDNEXT we deal with a partial solution vector  $x$  that satisfies  $x_i \in \{0, y_i\}$  for all  $i \in [n]$ . We call the (indices of the) non-zero components of the partial solution *fixed*. They indicate the components of the secret code that have already been identified. The (indices of the) zero components are called *open*. Whenever FINDNEXT makes a guess  $\sigma$ , it requires to know the number of open components in which the guess coincides with the secret code, i.e. the number

$$\text{black}(\sigma, y, x) := \text{black}(\sigma, y) - \text{black}(\sigma, x).$$

Note, that the term  $\text{black}(\sigma, x)$  is known by the codebreaker. After the first component of  $y$  has been found and fixed in  $x$ , there exists a  $j \in [n]$  such that  $\text{black}(\sigma^j, y, x) = 0$ . As long as we have open components in  $x$ , we can either find a  $j \in [n-1]$  with  $\text{black}(\sigma^j, y, x) > 0$  but  $\text{black}(\sigma^{j+1}, y, x) = 0$  and set  $r := j + 1$ , or we have  $\text{black}(\sigma^n, y, x) > 0$  but  $\text{black}(\sigma^1, y, x) = 0$  and set  $j := n$  and  $r := 1$ . Again, we call such an index  $j$  an *active* index. Let  $j$  be an active index and  $r$  its related index. Let  $c$  be the color of some component of  $y$  that is already identified and fixed in the partial solution  $x$ . With  $l_j$  and  $l_r$  we denote the position of color  $c$  in  $\sigma^j$  and  $\sigma^r$  respectively. The peg with color  $c$  serves as a pivot peg for identifying a correct position  $m$  in  $\sigma^j$  that is not fixed, yet. There are two possible modes for the binary search that depend on the fact if  $m \leq l_j$ . The mode is indicated by a Boolean variable leftS and determined by lines 4 to 8 of FINDNEXT. Clearly,  $m \leq l_j$  if  $l_j = n$ . Otherwise, the codebreaker guesses

$$\sigma^{j,0} := \left( c, (\sigma_i^j)_{i=1}^{l_j-1}, (\sigma_i^j)_{i=l_j+1}^n \right),$$

By the information  $\sigma_i^j = \sigma_{i+1}^r$  we obtain that  $(\sigma_i^j)_{i=1}^{l_j-1} \equiv (\sigma_i^r)_{i=2}^{l_j}$ . We further know that every open color has a wrong position in  $\sigma^r$ . For that reason,  $\text{black}(\sigma^{j,0}, y, x) = 0$  implies that  $m \leq l_j$ .

The binary search for the exact value of  $m$  is done in the interval  $[a, b]$ , where  $m$  is initialized as  $n$  and

---

**Algorithm 2:** Function FINDNEXT

---

**input** : Code  $y$ , partial solution  $x \neq 0$  and an active index  $j \in [n]$   
**output**: Position  $m$  of a correct open component in  $\sigma^j$

```
1 if  $j = n$  then  $r := 1$  ;
2 else  $r := j + 1$ ;
3 Choose a color  $c$  with identified position (a value  $c$  of some non-zero component of  $x$ );
4 Let  $l_j$  and  $l_r$  be the positions with color  $c$  in  $\sigma^j$  and  $\sigma^r$ , respectively;
5 if  $l_j = n$  then leftS := true;
6 else
7   Guess  $\sigma^{j,0} := (c, (\sigma_i^j)_{i=1}^{l_j-1}, (\sigma_i^j)_{i=l_j+1}^n)$ ;
8    $s := \text{black}(\sigma^{j,0}, y, x)$ ;
9   if  $s = 0$  then leftS := true;
10  else leftS := false;
11 if leftS then let  $a := 1$  and  $b := l_j$ ;
12 else let  $a := l_r$  and  $b := n$ ;
13  $m := n$  ; // position to be found
14 while  $b > a$  do
15    $l := \lceil \frac{a+b}{2} \rceil$  ; // position for peg  $c$ 
16   if leftS then  $\sigma^{j,l} := ((\sigma_i^j)_{i=1}^{l-1}, c, (\sigma_i^r)_{i=l+1}^{l_j}, (\sigma_i^j)_{i=l_j+1}^n)$ ;
17   else  $\sigma^{j,l} := ((\sigma_i^r)_{i=1}^{l_r-1}, (\sigma_i^j)_{i=l_r}^{l-1}, c, (\sigma_i^r)_{i=l+1}^n)$ ;
18   Guess  $\sigma^{j,l}$ ;
19    $s := \text{black}(\sigma^{j,l}, y, x)$ ;
20   if  $s > 0$  then
21      $b := l - 1$ ;
22   if  $b < m$  then let  $m := b$ ;
23   else  $a := l$ ;
24 Return  $m$ ;
```

---

$[a, b]$  as

$$[a, b] := \begin{cases} [1, l_j] & \text{if leftS} \\ [l_r, n] & \text{else} \end{cases}$$

(lines 9 to 11 of FINDNEXT). In order to determine if there is an open correct component on the left side of the current center  $l$  of  $[a, b]$  in  $\sigma^j$  we can define a case dependent permutation:

$$\sigma^{j,l} := \begin{cases} ((\sigma_i^j)_{i=1}^{l-1}, c, (\sigma_i^j)_{i=l}^{l_j-1}, (\sigma_i^j)_{i=l_j+1}^n) & \text{if leftS} \\ ((\sigma_i^r)_{i=1}^{l_r-1}, (\sigma_i^r)_{i=l_r+1}^l, c, (\sigma_i^r)_{i=l+1}^n) & \text{else} \end{cases}$$

In the first case, the first  $l - 1$  components of  $\sigma^{j,l}$  coincide with those of  $\sigma^j$ . The remaining components of  $\sigma^{j,l}$  cannot coincide with the corresponding components of the secret code if they have not been fixed, yet. This is because the  $l$ -th component of  $\sigma^{j,l}$  has the already fixed value  $c$ , components  $l + 1$  to  $l_j$  coincide with the corresponding components of  $\sigma^r$  which satisfies  $\text{black}(\sigma^r, y, x) = 0$  and the remaining components have been checked to be wrong in this case. Thus, there is a correct open component on the left side of  $l$  in  $\sigma^j$ , if and only if  $\text{black}(\sigma^{j,l}, y, x) \neq 0$ . In the second case, the same holds for similar arguments. Now, if there is a correct open component to the left of  $l$ , we update the binary search interval  $[a, b]$  by  $[a, l - 1]$  and set  $m := \min(m, l - 1)$ . Otherwise, we update  $[a, b]$  by  $[l, b]$ .

**The Main Algorithm.** The main algorithm is outlined as Algorithm 3. It starts with an empty

---

**Algorithm 3:** Algorithm for Permutations

---

```

1 Let  $y$  be the secret code and set  $x := (0, 0, \dots, 0)$ ;
2 Guess the permutations  $\sigma^i$ ,  $i \in [n-1]$ ;
3 Initialize  $v \in \{0, 1, \dots, n\}^n$  by  $v_i := \text{black}(\sigma^i, y)$ ,  $i \in [n-1]$ ,  $v_n := n - \sum_{i=1}^{n-1} v_i$ ;
4 if  $v = \mathbf{1}_n$  then
5    $j := 1$ ;
6   Find the position  $m$  of the correct peg in  $\sigma^1$  by at most  $\frac{n}{2} + 1$  further guesses;
7 else
8   Call FINDFIRST for an active  $j \in [n]$  to find the position of the correct peg in  $\sigma^j$  by at most
    $2\lceil \log_2 n \rceil$  further guesses;
9    $x_m := \sigma_m^j$ ;
10   $v_j := v_j - 1$ ;
11  while  $|\{i \in [n] \mid x_i = 0\}| > 2$  do
12    Choose an active index  $j \in [n]$ ;
13     $m := \text{FINDNEXT}(y, x, j)$ ;
14     $x_m := \sigma_m^j$ ;
15     $v_j := v_j - 1$ ;
16 Make at most two more guesses to find the remaining two unidentified colors;

```

---

partial solution and finds the components of the secret code  $y$  one-by-one. Herein, the vector  $v$  does keep record about the number of open components in which the permutations  $\sigma^1, \dots, \sigma^n$  equal  $y$  and is, thus, initialized by  $v_i := \text{black}(\sigma^i, y)$ ,  $i \in [n-1]$  and  $v_n := n - \sum_{i=1}^{n-1} v_i$ . As mentioned above, the main loop always requires an active index. For that reason, if  $v = \mathbf{1}_n$  in the beginning, we fix one solution peg in  $\sigma^1$  and update  $x$  and  $v$ , correspondingly. Every call of FINDNEXT in the main loop augments  $x$  by a correct solution value. Since one call of findNext requires at most  $1 + \lceil \log_2 n \rceil$  guesses, Algorithm 3 does not need more than  $(n-3)\lceil \log_2 n \rceil + \frac{5}{2}n - 1$  queries (inclusive at most  $\frac{n}{2} + 1$  initial and 2 final queries, respectively) to break the secret code.

**Example.** We consider the case  $n = k = 8$  and suppose that the secret code  $y$  is

7    1    4    3    2    8    5    6

Figure 1 shows  $n$  possible initial queries. We illustrate the procedure FINDNEXT and further suppose that

	queries								$n_1$	$n_2$	$n_3$
$\sigma^1$	1	2	3	4	5	6	7	8	0	0	0
$\sigma^2$	2	3	4	5	6	7	8	1	2	0	2
$\sigma^3$	3	4	5	6	7	8	1	2	3	2	1
$\sigma^4$	4	5	6	7	8	1	2	3	1	1	0
$\sigma^5$	5	6	7	8	1	2	3	4	0	0	0
$\sigma^6$	6	7	8	1	2	3	4	5	0	0	0
$\sigma^7$	7	8	1	2	3	4	5	6	1	0	1
$\sigma^8$	8	1	2	3	4	5	6	7	1	0	1

Figure 1: Initial queries  $\sigma^j$  with associated responses  $n_1 = \text{black}(\sigma^j, y)$ , coincidences with a partial solution  $n_2 = \text{black}(\sigma^j, x)$ , and the difference of both  $n_3$ .

	queries								$n_1$	$n_2$	$n_3$
$\sigma^a$	2	7	8	1	3	4	5	6	2	2	0
$\sigma^b$	7	8	2	1	3	4	5	6	3	2	1
$\sigma^c$	7	2	8	1	3	4	5	6	3	2	1

Figure 2: Binary search queries to extend the partial solution. The highlighted subsequences correspond to the subsequences of the selected initial queries.

we have already identified the positions of 3 colors indicated in the partial solution  $x$ :

•   •   •   •   2   •   5   6

From the  $n_3$  values in Figure 1 we see that  $\text{black}(\sigma_3, y, x) = 1$  and  $\text{black}(\sigma_4, y, x) = 0$ , so we choose 3 as our active index applying FINDNEXT with the highlighted initial queries,  $\sigma^3$  and  $\sigma^4$ . Choosing the already identified color 2 as a pivot color, FINDNEXT does its binary search to identify the next correct peg as demonstrated in Figure 2. Since the information  $n_3$  for query  $\sigma^a$  is 0 (cf. lines 5-7 of Algorithm 2) all correctly placed pegs in  $\sigma^3$  are on the left side of the pivot peg. Thus, we can apply a binary search for the left most correct peg in the first 4 places of query  $\sigma^3$  using the pivot peg. here, the binary search is done by queries  $\sigma^b$  and  $\sigma^c$  and identifies the peg with color 7 (in general, the peg that is left to the most left pivot position for which  $n_3$  is non-zero). If the response to  $\sigma^a$  would have been greater than 0, we would have found analogously a new correct peg in  $\sigma^3$  on the right side of the pivot peg.

## 2.2 The Case $k > n$

Now, we consider the variant of Black-Peg Mastermind where  $k > n$  and color repetition is forbidden. Let  $y = (y_1, \dots, y_n)$  be the code that must be found. We use the same notations as above.

**Phase 1.** Consider the  $k$  permutations  $\bar{\sigma}^1, \dots, \bar{\sigma}^k$ , where  $\bar{\sigma}^1$  corresponds to the identity map on  $[k]$  and for  $j \in [k-1]$ , we obtain  $\bar{\sigma}^{j+1}$  from  $\bar{\sigma}^j$  by a circular shift to the right. We define  $k$  codes  $\sigma^1, \dots, \sigma^k$  by  $\sigma^j = (\bar{\sigma}_i^j)_{i=1}^n$ ,  $j \in [k]$ . Within those  $k$  codes, every color appears exactly once at every position and, thus, we have

$$\sum_{j=1}^k \text{black}(\sigma^j, y) = n,$$

similar to (1). Since  $k > n$ , this implies that

**Lemma 1.** *There is a  $j \in [k]$  with  $\text{black}(\sigma^j, y) = 0$ .*

**Phase 2.** Having more colors than holes, we can perform our binary search for a next correct position without using a pivot peg. The corresponding simplified version of FINDNEXT is outlined as Algorithm 4. Using that version of FINDNEXT also allows to simplify our main algorithm (Algorithm 3) by adapting lines 2 and 3, and, due to Lemma 1, skipping lines 4-10. Thus, for the required number of queries to break the secret code we have: the initial  $k-1$  guesses, a call of the modified FINDNEXT for every but the last two positions (at most  $\lceil \log_2 n \rceil$  guesses per position) and one or two final guesses. This yields, that the modified Mastermind Algorithm breaks the secret code in at most  $(n-2)\lceil \log_2 n \rceil + k + 1$  queries.

---

**Algorithm 4:** Function FINDNEXT for  $k > n$ 

---

**input** : Code  $y$ , partial solution  $x \neq 0$  and an active index  $j \in [k]$   
**output**: Position  $m$  of a correct open component in  $\sigma^j$

```
1 if  $j = n$  then  $r := 1$  ;  
2 else  $r := j + 1$ ;  
3  $a := 1, b := n$ ;  
4  $m := 1$  ; // position to be found  
5 while  $b > a$  do  
6    $l := \lceil \frac{a+b}{2} \rceil$  ; // mid position of current interval  
7   Guess  $\sigma := ((\sigma_i^r)_{i=1}^{l-1}, (\sigma_i^j)_{i=l}^n)$ ;  
8    $s := \text{black}(\sigma, y, x)$ ;  
9   if  $s > 0$  then  
10     $a := l$ ;  
11    if  $a > m$  then let  $m := a$ ;  
12  else  $b := l - 1$ ;  
13 Return  $m$ ;
```

---

### 3 Lower Bounds on the Number of Queries

In the following we consider the case that the secret code has no repetition but arbitrary questions are allowed. Note that the lower bounds for that case especially hold true for AB-Mastermind and Permutation-Mastermind, respectively, since the codebreaker will not be able to detect a secret code with less attempts, if the set of allowed queries is restricted to the corresponding subset. Similar to the upper bounds, we proof the respective lower bounds on the necessary number of queries by construction.

#### 3.1 The Case $k = n$ : Permutation-Mastermind

Notice that the achieved bound for the case  $k = n$  especially holds for Permutation-Mastermind. In each iteration, the worst case for the code breaker is simulated by allowing the code maker to replace his secret code with another permutation from the remaining feasible search space. For  $m \in \mathbb{N}$  we denote the  $m$ -th query of the code breaker with  $x^m$  and the  $m$ -th secret code adaption of the code maker with  $y^m$ . The remaining feasible search space  $R_m$  consists of all permutations that agree with the first  $m$  pairs of queries and answers:

$$R_m := \{\sigma \in S_n \mid \forall j \in [m] : \text{black}(y^j, x^j) = \text{black}(\sigma, x^j)\}.$$

Now, a simple strategy of the code maker is to reply every query  $x^m$ ,  $m \in \mathbb{N}$ , with the smallest possible number

$$b_m := \min_{\sigma \in R_{m-1}} \text{black}(\sigma, x^m),$$

choosing his new secret code  $y^m \in R_{m-1}$  such that  $\text{black}(y^m, x^m) = b_m$ . We obtain our lower bound on the necessary number of queries by proving the following

**Lemma 2.** *It holds that  $b_m \leq m$  for all  $m \in \mathbb{N}$ .*

In particular, non of the first  $n - 1$  queries will be answered with  $n$ . Thus, the secret code can not be identified with less than  $n$  queries.



*Proof.* Assuming that our claim is wrong, we fix the smallest number  $m \in [n]$  with  $b_m > m$ . Let

$$D := \{c \in [n] \mid (x^m)^{-1}(c) = (y^m)^{-1}(c)\}$$

be the set of colors that are correctly placed in the current query with respect to the current secret code. For every  $i \in [n]$  let  $C_i \subseteq [n]$  be the set of all colors that do not occur at position  $i$  in any of the former  $m - 1$  queries nor in the current secret code, i.e.,

$$C_i := \{c \in [n] \mid c \neq x^\ell(i) \text{ for all } \ell \in [m]\}.$$

The intersections  $C_i \cap D$ ,  $i \in [n]$ , are not empty since  $|D| = b_m \geq m + 1$  but at most  $m$  of the  $n$  colors are missing in  $C_i$ . This fact will enable us to determine a new feasible secret code  $z \in R_{m-1}$  such that  $\text{black}(z, x^j) = b_j$  for all  $j \in [m - 1]$  but  $\text{black}(z, x^m) < b_m$ , a contradiction to the minimality of  $b_m$ . The new secret code  $z$  is constructed from  $y^m$  by changing the colors of some components that coincide with  $x^m$ , choosing the new color at a given position  $i$  from  $C_i \cap D$ . The precise procedure is outlined as Algorithm 5. Starting with any position  $i_1$  where  $y^m$  and  $x^m$  have the same color, we choose another

---

**Algorithm 5:** Secret code adaption,  $k = n$

---

```

1 Set  $s := 1$  and  $A := \emptyset$ ;
2 Choose position  $i_1 \in [n]$  with  $y^m(i_1) = x^m(i_1)$ ;
3 Choose color  $c_1 \in C_{i_1} \cap D$ ;
4 while  $c_s \notin A$  do
5    $A := A \cup \{y^m(i_s)\}$ ;
6    $s := s + 1$ ;
7   Define position  $i_s := (y^m)^{-1}(c_{s-1})$ ;
8   Choose color  $c_s \in C_{i_s} \cap D$ ;
9 Find the unique  $t < s$  with  $y^m(i_t) = c_s$ ;
10  $z := y^m$ ;
11 for  $\ell := t$  to  $s$  do  $z(i_\ell) := c_\ell$ 

```

---

color  $c_1 \in C_{i_1} \cap D$ . Since  $c_1 \in D$ , there must be another position  $i_2$  such that  $y^m(i_2) = c_1 = x^m(i_2)$ . Thus, for  $s > 1$  we can iteratively determine positions  $i_s$  where  $y^m$  and  $x^m$  have the same color,  $c_{s-1}$ , and choose a new color  $c_s \in C_{i_s} \cap D$  (While loop, lines 4–8). The iteration stops, if the chosen color  $c_s$  corresponds with a color that appears in  $y^m$  at some position  $i_t$ ,  $t < s$ , that has been considered before (indicated by the set  $A$ ). Note, that the iteration must terminate with  $2 \leq s \leq m + 1$ , since  $A$  is empty in the beginning, and  $|D| = m + 1$ . The set of chosen colors  $\{c_\ell \mid t \leq \ell \leq s\}$  is equal to the set of colors  $\{y^m(i_\ell) \mid t \leq \ell \leq s\}$  at the corresponding positions in  $y^m$ . Hence, the new secret code  $z$  (defined in lines 10–11) is again a permutation. Now, let  $j \in [m - 1]$  be the number of some former query. Due to the minimal choice of  $m$  we have  $\text{black}(z, x^j) \geq b_j$ . But  $\text{black}(z, x^j) \leq b_j$  does also hold since  $\text{black}(y^m, x^j) = b_j$  ( $y^m \in R_m$ ) and for each position  $i$  with  $z(i) \neq y^m(i)$  we have  $z(i) \neq x^j(i)$  ( $z(i) \in C_i$ ). Further, the construction of  $z$  immediately yields that  $\text{black}(z, x^m) < b_m$ . Thus,  $z$  is indeed a secret permutation in  $R_{m-1}$  that contradicts the minimality of  $b_m$ .  $\square$

### 3.2 The Case $k > n$

Considering the case  $k > n$  we adapt the code maker strategy from the former subsection, i.e. in each turn  $m$  the code maker chooses the new secret code  $y^m$  such that the answer is the smallest possible answer  $b_m$ . We easily obtain a lower bound of  $k$  queries by the following

**Lemma 3.** *It holds that  $b_m < n$  for all  $m < k$ .*

*Proof.* Assume for a moment that there exists an  $m < k$  with  $b_m = n$ . Like before, let

$$C_i := \{c \in [n] \mid c \neq x^\ell(i) \text{ for all } \ell \in [m]\}.$$

Similar to Algorithm 5 we now replace certain entries of  $y^m$  by elements of the corresponding  $C_i$ . The detailed procedure is described in Algorithm 6.

---

**Algorithm 6:** Secret code adaption,  $k > n$

---

```

1 Set  $s := 1$  and  $i_1 := 1$ ;
2 Set  $A := \emptyset$  and  $B := \{c \in [k] \mid \forall i \in [n] : y^k(i) \neq c\}$ ;
3 Choose color  $c_1 \in C_1$ ;
4 while  $c_s \notin A \cup B$  do
5    $A := A \cup \{y^m(i_s)\}$ ;
6    $s := s + 1$ ;
7   Define position  $i_s := (y^m)^{-1}(c_{s-1})$ ;
8   Choose color  $c_s \in C_{i_s}$ ;
9 if  $c_s \in A$  then
10  Find the unique  $t < s$  with  $y^m(i_t) = c_s$ ;
11 else
12  Set  $t := 1$ ;
13  $z := y^m$ ;
14 for  $\ell := t$  to  $s$  do  $z(i_\ell) := c_\ell$ 
```

---

We start with position one and choose a color  $c_1 \in C_{i_1}$ . As soon as we have  $c_s \in B$ , we construct  $z$  by starting with  $y^k$  and then replacing the color  $y^m(i_\ell)$  by the color  $c_{i_\ell}$  for any  $\ell \leq s$ . The set of chosen colors  $\{c_\ell \mid \ell \leq s\}$  is equal to the set of colors  $\{y^m(i_\ell) \mid \ell \leq s\}$  except for  $c_s$  which only appears in the first set and  $y^m(i_\ell)$  which only appears in the second. Since  $c_s \in B$  we know that  $z$  has no color occurring twice.

If the Iteration stops because of  $c_s \in A$  the procedure is identic to the one in Algorithm 5. So in both cases we find that  $\text{black}(z, x^m) < b_m$  and  $\text{black}(z, x^\ell) = b_\ell$  for any  $\ell \in [m - 1]$ , in contradiction to the minimality of  $b_m$ .  $\square$

## 4 Conclusions and Further Work

In this paper we presented a deterministic algorithm for the identification of a secret code in “Permutation Mastermind” and “Black-Peg AB-Mastermind” with more colors than positions. A challenge of these Mastermind variants is that no color repetition is allowed for a query while most strategies for other Mastermind variants exploit the property of color repetition. Furthermore we improved the recent lower bound of Berger et al. [1] and showed that the worst case number of queries for Permutation Mastermind is at least  $n$ , another matter than the asymptotic bound of  $O(n)$ , which is long-established. Ko and Teng [18] conjecture that this number is actually  $\Omega(n \log n)$ , a proof of which would close the gap to the upper bound. The lower bound proof of Berger et al. is derived by solely considering the search space partition with respect to the number of coincidences with the very first query. On the other hand, our algorithmic proof does not exploit any structure property of the remaining search space. For both reasons we expect at least some room for improvements of the lower bound. In the future we will take both bounds in focus but the real challenge is to prove or disprove the conjecture of Ko and Teng.

## References

- [1] A. Berger, C. Chute, and M. Stone. Query Complexity of Mastermind Variants. *arXiv:1607.04597 [math.CO]*, 2016.
- [2] L. Berghman, D. Goossens, and R. Leus. Efficient solutions for Mastermind using genetic algorithms. *Computers & OR*, 36(6):1880–1885, 2009.
- [3] Z. Chen, C. Cunha, and S. Homer. Finding a Hidden Code by Asking Questions. In: *Proceedings of the 2nd Conference on Computing and Combinatorics (COCOON 1996)*, pages 50–56. Springer, 1996.
- [4] Vasek Chvátal. Mastermind. *Combinatorica*, 3:325–329, 1983.
- [5] B. Doerr, R. Spöhel, H. Thomas, and C. Winzen. Playing Mastermind with Many Colors. In: *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*, pages 695–704. SIAM Society for Industrial and Applied Mathematics, 2013.
- [6] B. Doerr and C. Winzen. Playing Mastermind with Constant-Size Memory. In: *Proceedings of the Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, pages 441–452, 2012.
- [7] P. Erdős and C. Rényi. On Two Problems in Information Theory. *Publications of the Mathematical Institute of the Hungarian Academy of Science*, 8:229–242, 1963.
- [8] M. El Ouali and V. Sauerland. Improved Approximation Algorithm for the Number of Queries Necessary to Identify a Permutation. *arXiv:1303.5862v2 [cs.DS]*, 2013.
- [9] R. Focardi and F. L. Luccio. Cracking Bank PINs by Playing Mastermind. In: *Proceedings of the 5th International Conference on Fun with Algorithms (FUN 2010)*, pages 202–213. Springer, 2010.
- [10] J. J. M. Guervós, C. Cotta, and A. M. Gacia. Improving and Scaling Evolutionary Approaches to the Mastermind Problem. In: *Proceedings of Applications of Evolutionary Computation (EvoApplications 2011)*, pages 103–112. Springer, 2011.
- [11] J. J. M. Guervós, A. M. Mora, and C. Cotta. Optimizing worst-case scenario in evolutionary solutions to the Mastermind puzzle. In: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2011)*, pages 2669–2676. IEEE, 2011.
- [12] M. T. Goodrich. The Mastermind Attack on Genomic Data. In: *Proceedings of the 30th IEEE Symposium on Security and Privacy (SP 2009)*, pages 204–218. IEEE, 2009.
- [13] M. T. Goodrich. On the algorithmic complexity of the Mastermind game with black-peg results. *Information Processing Letters*, 109:675–678, 2009.
- [14] G. Jäger and M. Peczarski. The number of pessimistic guesses in Generalized Mastermind. *Information Processing Letters*, 109:635–641, 2009.
- [15] G. Jäger and M. Peczarski. The number of pessimistic guesses in Generalized Black-peg Mastermind. *Information Processing Letters*, 111:933–940, 2011.
- [16] G. Jäger and M. Peczarski. The worst case number of questions in Generalized AB game with and without white-peg answers. *Discrete Applied Mathematics*, 184:20–31, 2015.
- [17] D. E. Knuth. The computer as a master mind. *Journal of Recreational Mathematics*, 9:1–5, 1977.
- [18] K. Ko and S. Teng. On the Number of Queries Necessary to Identify a Permutation. *Journal of Algorithms*, 7:449–462, 1986.
- [19] T. Kalisker and D. Camens. Solving Mastermind Using Genetic Algorithms. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, pages 1590–1591. ACM, 2003.
- [20] K. Koyama and T. W. Lai. An optimal Mastermind strategy. *Journal of Recreational Mathematics*, 25:251–256, 1993.

- [21] J. Stuckman and G. Zhang. Mastermind is  $\mathcal{NP}$ -Complete. *arXiv:cs/0512049v1 [cs.CC]*, 2005.
- [22] G. Viglietta. Hardness of Mastermind. In: *Proceedings of the 6th International Conference on Fun with Algorithms (FUN 2012)*, pages 368–378. Springer, 2012.